

Doxxygen_MPDASTAR_src_DB Reference Manual

Generated by Doxygen 1.3.7

Wed Sep 6 20:08:50 2006

Contents

| | | |
|----------|---|----------|
| 1 | Doxygen_MPDSrcDB Class Index | 1 |
| 1.1 | Doxygen_MPDSrcDB Class List | 1 |
| 2 | Doxygen_MPDSrcDB File Index | 3 |
| 2.1 | Doxygen_MPDSrcDB File List | 3 |
| 3 | Doxygen_MPDSrcDB Class Documentation | 5 |
| 3.1 | time_keeper Struct Reference | 5 |
| 4 | Doxygen_MPDSrcDB File Documentation | 7 |
| 4.1 | boo.c File Reference | 7 |
| 4.2 | eventCnts.c File Reference | 8 |
| 4.3 | insTrgCnt.h File Reference | 14 |
| 4.4 | N_insTrgCnt.cc File Reference | 22 |
| 4.5 | resync.c File Reference | 31 |
| 4.6 | resync.h File Reference | 36 |
| 4.7 | syncCol.c File Reference | 39 |
| 4.8 | syncCol.h File Reference | 45 |
| 4.9 | test.c File Reference | 49 |
| 4.10 | time_keeper.c File Reference | 51 |
| 4.11 | time_keeper.h File Reference | 53 |

Chapter 1

Doxxygen_MPDU_STAR_src_DB Class Index

1.1 Doxygen_MPDU_STAR_src_DB Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

| | |
|---------------------------------------|---|
| time_keeper | 5 |
|---------------------------------------|---|

Chapter 2

Doxxygen_MP_D_STAR_src_DB File Index

2.1 Doxygen_MP_D_STAR_src_DB File List

Here is a list of all files with brief descriptions:

| | |
|--------------------------|----|
| boo.c | 7 |
| eventCnts.c | 8 |
| insTrgCnt.h | 14 |
| N_insTrgCnt.cc | 22 |
| resync.c | 31 |
| resync.h | 36 |
| syncCol.c | 39 |
| syncCol.h | 45 |
| test.c | 49 |
| time_keeper.c | 51 |
| time_keeper.h | 53 |

Chapter 3

Doxxygen_MPDU_STARD_src_DB Class Documentation

3.1 time_keeper Struct Reference

Public Attributes

- `clock_t begin_clock`
- `clock_t save_clock`
- `time_t begin_time`
- `time_t save_time`

3.1.1 Member Data Documentation

3.1.1.1 `clock_t time_keeper::begin_clock`

Definition at line 13 of file time_keeper.c.

3.1.1.2 `time_t time_keeper::begin_time`

Definition at line 14 of file time_keeper.c.

3.1.1.3 `clock_t time_keeper::save_clock`

Definition at line 13 of file time_keeper.c.

3.1.1.4 `time_t time_keeper::save_time`

Definition at line 14 of file time_keeper.c.

The documentation for this struct was generated from the following file:

- `time_keeper.c`

Chapter 4

Doxxygen_MPDUSTAR_src_DB File Documentation

4.1 boo.c File Reference

```
#include <stdio.h>
#include <string.h>
```

4.2 eventCnts.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include <iostream>
#include "resync.h"
#include "time_keeper.h"
```

Functions

- int **main** (int argc, char *argv[])
 - int **rowCmp** (MYSQL_ROW str1, MYSQL_ROW str2, MYSQL_RES *res)
 - void **printRow** (MYSQL_ROW row, MYSQL_RES *res, int flag)
 - void **printPrimaryKeys** (MYSQL_RES *res)
 - void **help** ()
 - void **cleanUp** (MYSQL *dbH, MYSQL *dbS, int value)

4.2.1 Function Documentation

4.2.1.1 void cleanUp (MYSQL * *dbH*, MYSQL * *dbS*, int *value*)

Definition at line 268 of file eventCnts.c.

```
269 {
270     switch (value)
271     {
272         case 1:
273             printf( "\nntables are synchronized \n\n");
274             prn_time();
275             break;
276         case 2:
277             printf ( "\nntables are NOT synchonized \n\n");
278             prn_time();
279             break;
280         default:
281             printf ("SOMETHING IS VERY WRONG! (cleanUp() switch default) \n");
282     }
283
284     mysql_close(dbH);
285     mysql_close(dbS);
286
287     exit(1);
288 }
```

4.2.1.2 void help()

Definition at line 261 of file eventCnts.c.

```

262 {
263   printf("HELP\n\tresync checks two tables from two different servers (a master and a slave) to \n\t1")
264   printf("\nUSAGE:\n\t>resync databaseName host1(master) port user table host2(slave) port\nFor example"
265
266 }

```

4.2.1.3 int main (int argc, char * argv[])

Definition at line 17 of file eventCnts.c.

```

18 {
19   if ((argc < 2 || strcmp(argv[1],"--help")==0 ))
20   {
21     help();
22     return(1);
23   }
24 /*   if (argc < 8) */
25 /*   { */
26 /*     printf("usage is incorrect - use --help for correct usage\n"); */
27 /*     return(1); */
28 /*   } */
29
30
31 /*   char *db = argv[1]; */
32 /*   char *mhost = argv[2]; */
33 /*   int mport = atoi(argv[3]); */
34 /*   char *user = argv[4]; */
35 /*   char *table = argv[5]; */
36 /*   char *shost = argv[6]; */
37 /*   int sport = atoi(argv[7]); */
38
39 /*   if ( !strcmp(argv[7],"3336" ) ) */
40 /*   { */
41 /*     pass = "mirrOr"; */
42 /*   } */
43 /*   */
44   char *host = "onldb.starp.bnl.gov";
45   char *db1 = "RunLog";
46   char *db2 = "RunLog_daq";
47   int port1 = 3501;
48   int port2 = 3503;
49   char *user = "deph";
50   char * pass = NULL;
51
52   char queryOne[40];
53   char queryTwo[255];
54   char queryAdd[155];
55   char orderBy[155];
56   char orderAdd[155];
57   int val = 1;
58
59   printf("\nFOR TABLE %s\n",table);
60   if (!mysql_real_connect(&dbH,host,user,pass,db1,port1,NULL,0))
61   {
62     printf("can't connect to %s %s %s\n",db,mhost,mport);
63   }
64
65   if (!mysql_real_connect(&dbS,host,user,pass,db2,port2,NULL,0))
66   {
67     printf("can't connect to %s %s %s\n",db,shost,sport);
68   }
69
70
71   sprintf(queryOne,"SELECT runNumber, eventCounts from daqSummary");

```

```

72     if (mysql_real_query (&dbH, queryOne, strlen(queryOne)))
73     {
74         printf("THIS query failed: %s\n%s\n",queryOne,mysql_error(&dbH));
75
76     }
77     res=mysql_store_result(&dbH);
78
79
80     int max = mysql_num_fields(res);
81     char* fieldList[max];
82     char* fieldNum[max];
83
84     // initialize to all 0s
85     for( int k = 0 ; k<max; k++)
86     {
87         fieldList[k] = 0;
88         fieldNum[k] = 0;
89     }
90     //GET the primary Keys separate them based on type
91     int j =0;
92         field = mysql_fetch_fields (res);
93     for (int i = 0; i<mysql_num_fields(res) ; i++)
94     {
95         field = mysql_fetch_field (res);
96
97         if(IS_PRI_KEY(field->flags)){
98             if(IS_NUM(field->type)){
99                 fieldNum[j] = field->name;
100                 // printf("FIELD NUM NAME IS %s \n",field->name);
101             }else{
102                 // printf("FIELD NAME IS %s \n",field->name);
103                 fieldList[j]= field->name;
104             }
105             j++;
106         }
107     }
108
109 //GET THE NUMBER OF PRIMARY KEYS
110     int count = 0, numbers =0, characters = 0;
111     for (int m = 0; m < max; m++)
112     {
113         if (fieldList[m]) {count++;}
114         if (fieldNum[m]) {count++;}
115     }
116 //strcat queryTwo here
117     sprintf(queryTwo,"SELECT ");
118     sprintf(orderBy, " ORDER BY ");
119
120     for (int n = 0; n < count; n++)
121     {
122         if (n < count-1)
123         {
124             if (fieldList[n]){
125                 sprintf(queryAdd,"%s, ",fieldList[n]);
126                 strcat(queryTwo,queryAdd);
127             }else{
128                 sprintf(queryAdd,"%s, ",fieldNum[n]);
129                 sprintf(orderAdd,"%s, ",fieldNum[n]);
130                 strcat(orderBy,orderAdd);
131                 strcat(queryTwo,queryAdd);
132             }
133         }else{
134             if (fieldList[n]){
135                 sprintf(queryAdd,"%s FROM %s", fieldList[n],table);
136                 strcat(queryTwo,queryAdd);
137             }else{
138                 sprintf(queryAdd,"%s FROM %s", fieldNum[n],table);

```

```

139                     sprintf(orderAdd,"%s",fieldNum[n]);
140                     strcat(orderBy,orderAdd);
141                     strcat(queryTwo,queryAdd);
142                 }
143             }
144         }
145         strcat(queryTwo,orderBy);
146 //        printf("QUERY IS %s\n",queryTwo);
147
148     //GET ALL KEYS FROM MASTER TABLE
149
150     start_time();
151     if (mysql_real_query (&dbH, queryTwo, strlen(queryTwo)))
152     {
153         printf("THIS query failed: %s\n%s\n",queryTwo,mysql_error(&dbH));
154
155     }else{
156         resM=mysql_store_result(&dbH);
157     }
158
159     if(resM)
160     {
161         printf("\nNUM OF ROWS FROM %s IS = %d\n\n",mhost,mysql_num_rows(resM));
162         //GET ALL KEYS FROM SLAVE TABLE
163         prn_time();
164         if (mysql_real_query (&dbS, queryTwo, strlen(queryTwo)))
165         {
166             printf("THIS query failed: %s\n%s\n",queryTwo,mysql_error(&dbS));
167
168         }else{
169
170             resS=mysql_store_result(&dbS);
171
172             if(resS)
173             {
174                 printf("\nNUM OF ROWS FROM %s IS = %d\n\n",shost,mysql_num_rows(resS));
175                 prn_time();
176                 printPrimaryKeys(resS);
177
178                 int cmp=0;
179
180                 while(1){
181                     if (cmp<=0)
182                     {
183                         rowM = mysql_fetch_row(resM);
184                     }
185                     if (cmp >=0)
186                     {
187                         rowS = mysql_fetch_row(resS);
188                     }
189                     if (rowM == 0 && rowS == 0) break;
190                     cmp = rowCmp(rowM,rowS,resS);
191                     if (cmp>0)
192                     {
193                         printRow(rowS,resS,1);
194                         printf("is missing from %s\n",mhost);
195                         val=2;
196                     }
197                     if (cmp<0)
198                     {
199                         printRow(rowM, resM,1);
200                         printf("is missing from %s\n",shost);
201                         val=2;
202                     }
203
204                 }
205             }else{
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
626
627
628
628
629
629
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
15
```

```

206                         printf("NO DATA RETURNED FOR %s\n",table);
207                     }
208                 }
209             }
210         cleanUp(&dbH,&dbS,val);
211     }
212 }
```

4.2.1.4 void printPrimaryKeys (MYSQL_RES *res)

Definition at line 248 of file eventCnts.c.

```

249 {
250     printf("primary keys are:\n\n");
251     for(int x = 0; x<mysql_num_fields(res); x++)
252     {
253         field2 = mysql_fetch_field(res);
254         if(IS_PRI_KEY(field2->flags))
255         {
256             printf("****%s***\t\t",field2->name);
257         }
258     }
259     printf("\n");
260 }
```

4.2.1.5 void printRow (MYSQL_ROW row, MYSQL_RES *res, int flag)

Definition at line 237 of file eventCnts.c.

```

238 {
239     //PRINT OUT A ROW FROM a RESULT SET
240     int num = mysql_num_fields(res);
241
242 for (int i=0 ; i<mysql_num_fields(res) ; i++)
243 {
244     // printf("\t%s",field->name );
245     printf("\t***%s***\n",row[i] );
246 }
247 }
```

4.2.1.6 int rowCmp (MYSQL_ROW str1, MYSQL_ROW str2, MYSQL_RES *res)

Definition at line 214 of file eventCnts.c.

```

215 {
216     if (str1 == 0) return 1;
217     if (str2 == 0) return -1;
218
219     for (int i=0; i<mysql_num_fields(resM);i++)
220     {
221
222         if(IS_NUM(field->type)){
223             float a = atof(str1[i]);
224             float b = atof(str2[i]);
225             float ans = a - b;
226             if (ans) return ans;
227         }else{
```

```
228     int ans = strcmp(str1[i],str2[i]);
229     if (ans) return ans;
230   }
231 }
232 }
233 return 0;
234
235 }
```

4.3 insTrgCnt.h File Reference

Functions

- int [tCounter](#) (int, MYSQL *, MYSQL *, MYSQL *, int)
- int [rundone](#) (int, MYSQL *, MYSQL *)
- int [getX](#) (MYSQL *, char *)
- int [updateFin](#) (MYSQL *, int, int)
- int [checkTags](#) (MYSQL *, int)
- void [cleanUp](#) (MYSQL *, MYSQL *, MYSQL *, int, int)

Variables

- MYSQL_RES * [res](#)
- MYSQL_RES * [res2](#)
- MYSQL_ROW [row](#)
- int [offlineBit](#)
- int [trgVersion](#)
- int [threshVersion](#)
- int [idx_trigger](#)
- int [fileSequence](#)
- int [fileStream](#)
- int [tCount](#)
- char * [name](#)
- int [count](#) = 0
- char [buff](#) [255]

4.3.1 Function Documentation

4.3.1.1 int checkTags (MYSQL *, int)

Definition at line 428 of file N_insTrgCnt.cc.

```

429 {
430     int event = 0;
431     int runTag=0;
432     char query5[255];
433     char query6[255];
434     // MYSQL_RES *res;
435     //MYSQL_ROW row;
436
437     sprintf (query5,"select run from daqEventTag where run = %d",run);
438     sprintf (query6,"select run from daqRunTag where run = %d",run);
439
440     if (mysql_real_query (db, query5, strlen(query5)))
441     {
442         printf("check 1 query failed %s\n",query5);
443         event=0;
444     }else{
445         res = mysql_store_result(db);
446         row = mysql_fetch_row(res);
447         if(row)
448             event=1;
449     }
450 }
```

```

451     mysql_free_result(res);
452
453
454     if (mysql_real_query (db, query6, strlen(query6)))
455     {
456         int erno = mysql_errno(db);
457         const char* err = mysql_error(db);
458         printf("check 2 query failed %s \t%d\t%s\n",query6,erno,err);
459         runTag=0;
460     }else{
461
462
463     res = mysql_store_result(db);
464     row = mysql_fetch_row(res);
465     if(row)
466         runTag=1;
467     }
468
469     mysql_free_result(res);
470
471     // printf("runtag = %d and event =%d, run = %d \n",runTag,event,run);
472     if ((runTag && event))
473         return 1;
474     else{ return 0;}
475 }
```

4.3.1.2 void cleanUp (MYSQL *, MYSQL *, MYSQL *, int, int)

Definition at line 477 of file N_insTrgCnt.cc.

```

478 {
479
480     switch (value)
481     {
482         case 1:
483             printf("%d: \t All runs are finished \n", time(NULL));
484             break;
485         case 2:
486             printf("%d: \t end.fin updated to 3, run %d will be rechecked\n", time(NULL),run);
487             break;
488         case 3:
489             printf("%d: \t end.fin updated to 1, with triggerCount for run %d\n",time(NULL),run);
490             break;
491         case 4:
492             printf("%d: \t end.fin updated to 2 for run %d\n",time(NULL),run);
493             break;
494         case 5:
495             printf("%d: \t end.fin update to 1,with NO triggerCount update for run %d\n",time(NULL),run);
496             break;
497         case 6:
498             printf("%d: \t General Query Error ^^^^^^message^^^^^ %d\n",time(NULL),run);
499             break;
500         case 7:
501             printf("%d: \t end.fin RESET to 0 %d\n",time(NULL),run);
502             break;
503         case 8:
504             printf("%d: \t end.fin updated to 2, No recorded RUNS, No trigger Updates for run %d\n",time(NULL));
505             break;
506         default:
507             printf("BAD STUFF");
508     }
509
510     mysql_close(db_conditions);
```

```

512     mysql_close(db_runlog);
513     mysql_close(db_trigger);
514
515     exit(1);
516 }
```

4.3.1.3 int getX (MYSQL *, char *)

Definition at line 349 of file N_insTrgCnt.cc.

```

350 {
351     // MYSQL_RES *res;
352     //MYSQL_ROW row;
353     int x;
354     if(mysql_real_query(db, query, strlen(query))) {
355         printf("Error executing query: %s\n",query);
356         return(0);
357     }
358     res = mysql_store_result(db);
359     if(res == NULL) {
360         printf("Error storing (%s) for query %s\n",mysql_error(db),query);
361         return(0);
362     }
363     row = mysql_fetch_row(res);
364     if(row){
365         x= atoi(row[0]);
366     }else{ return 0; }
367     mysql_free_result(res);
368
369     return x;
370 }
```

4.3.1.4 int rundone (int, MYSQL *, MYSQL *)

Definition at line 267 of file N_insTrgCnt.cc.

```

268 {
269
270 static char *dbname = "Conditions_rts";
271 static char *cquery = "select offlineBit, name from triggers where idx_rn=";
272
273     int dbdone=0;
274
275     int nevts;
276     int nevts_et;
277     int end_testcount;
278     int nevbs;
279     int dest;
280     int ntags;
281
282     char query[255];
283
284     // check config to get list of evbs
285     // check end_test run record
286     // get count of events from evbs
287     // get count of events from eventTag
288     sprintf(query, "select count(*) from end where idx_rn=%d",run);
289     end_testcount = getX(db_conditions, query);
290
291     if(end_testcount == 0) {
292         printf("No end_test run record\n");
```

```

293     dbdone = 0;
294     goto done;
295 }
296
297 sprintf(query, "select count(*) from nodes where task=20 and idx_rn=%d",run);
298 nevbs = getX(db_conditions, query);
299
300 if(nevbs == 0) {
301     printf("No evbs in run %d.  Dbs as done as they will get.\n",run);
302     dbdone = 1;
303     goto done;
304 }
305
306 // Get the data destination
307
308 sprintf(query, "select destination from run where idx_rn=%d",run);
309 dest = getX(db_conditions, query);
310
311 if((dest != 2) &&
312     (dest != 4) &&
313     (dest != 3)) {
314
315     printf("No db records will be written for this run, run %d (data dest=%d)\n",run,dest);
316     dbdone = 1;
317     goto done;
318 }
319
320 sprintf(query, "select count(*) from daqRunTag where whichtag=1 and run=%d",run);
321 ntags = getX(db_runlog, query);
322
323 if(ntags != nevbs) {
324     printf("The number of run tags for run %d does not match the number of evbs %d vs %d\n", run, ntags, nevbs);
325     dbdone = 0;
326     goto done;
327 }
328
329 sprintf(query, "select sum(totalNumberOfEvents) from daqRunTag where run=%d\n",run);
330 nevts = getX(db_runlog, query);
331
332 sprintf(query, "select count(*) from daqEventTag where run=%d\n",run);
333 nevts_et = getX(db_runlog, query);
334
335 if(nevts != nevts_et) {
336     printf("Run tag and event tag disagree on number of events in run %d: %d vs %d\n",
337             run, nevts, nevts_et);
338     dbdone = 0;
339     goto done;
340 }
341
342 dbdone = 1;
343 done:
344
345 printf(dbdone ? "Run DBs are done\n" : "Run DBs are not done\n");
346 return(dbdone);
347 }

```

4.3.1.5 int tCounter (int, MYSQL *, MYSQL *, MYSQL *, int)

Definition at line 172 of file N_insTrgCnt.cc.

```

173 {
174     int val;
175     int updated;
176     int updated2;

```

```

177     char query[255];
178     char query2[255];
179     char query3[255];
180
181
182     sprintf(query,"SELECT name, trgVersion, threshVersion, offlineBit, idx_trigger from triggers where
183             if (mysql_real_query (db_conditions, query, strlen(query)))
184             {
185                 printf("THIS query failed: %s\n%s\n",query,mysql_error(db_conditions));
186                 return(6);
187             }else{
188                 res= mysql_store_result(db_conditions);
189                 if (res)
190                 {
191                     while (( row = mysql_fetch_row (res)) !=NULL)
192                     {
193                         for (int i = 0; i< mysql_num_fields(res); i++)
194                         {
195                             name = row[0];
196                             trgVersion = atoi(row[1]);
197                             threshVersion = atoi(row[2]);
198                             offlineBit = atoi(row[3]);
199                             idx_trigger = atoi(row[4]);
200                         }
201                     sprintf(query2,"SELECT count(*), fileStream, fileSequence from daqEventTag where
202                         (void) mysql_real_escape_string (db_runlog,buff,strlen(query2));
203
204                     if (mysql_real_query (db_runlog, buff, strlen(buff)))
205                     {
206                         printf("query2 failed: %s\n(%s)\n",query2,mysql_error(db_runlog));
207                         return(6);
208                     }
209                     res2=mysql_store_result(db_runlog);
210                     if (res2)
211                     {
212                         while ((row = mysql_fetch_row (res2)) !=NULL)
213                         {
214                             for (int j = 0; j < mysql_num_fields (res2);j++)
215                             {
216                                 tCount = atoi(row[0]);
217                                 fileStream = atoi(row[1]);
218                                 fileSequence = atoi(row[2]);
219                             }
220                         sprintf(query3,"INSERT into triggerCount set name = '%s', trgVersion =
221                         if (mysql_real_query (db_trigger, query3, strlen(query3)))
222                         {
223                             printf("insert failed query = %s\n%s\n",query3,mysql_error(db_trig
224                             return(6);
225
226                         }else{
227                             count++;
228                         }
229                     }
230                 }else{
231                     printf("INNER query returned nothing\n");
232                     return(6);
233                 }
234             }
235
236
237         }else{
238             if(!two){
239                 updated = updateFin(db_conditions, run,1);
240             }else{
241                 updated = updateFin(db_conditions, run,2);
242             }
243             if(updated){

```

```

244             return(5);
245         }else{
246             printf("\nupdate failed for run %d\n", run);
247             return(6);
248         }
249     }
250     if(!two){
251         updated2 = updateFin(db_conditions, run,1);
252     }else{
253         updated2 = updateFin(db_conditions, run,2);
254     }
255     if(updated2)
256     {
257         printf("triggerCount updated with %d inserts \n",count);
258         return(3);
259     }else{
260         printf("update failed for run %d\n%s\n", run,mysql_error(db_conditions));
261         return(6);
262     }
263 }
264 }
```

4.3.1.6 int updateFin (MYSQL *, int, int)

Definition at line 372 of file N_insTrgCnt.cc.

```

373 {
374     char query4[255];
375     char query7[255];
376     char query8[255];
377     char query9[255];
378
379     switch (val)
380     {
381     case 1:
382         sprintf(query4,"UPDATE end set entryTime = entryTime, fin = 1 where idx_rn = %d",run);
383         if (mysql_real_query (db, query4, strlen(query4)))
384         {
385             printf("updatefailed query: %s",query4);
386             return(0);
387         }else{
388             return(1);
389         }
390         break;
391     case 2:
392         sprintf(query7,"UPDATE end set entryTime = entryTime, fin = 2 where idx_rn = %d",run);
393         if (mysql_real_query (db, query7, strlen(query7)))
394         {
395             printf("updatefailed query: %s",query7);
396             return(0);
397         }else{
398             return(1);
399         }
400         break;
401     case 3:
402         sprintf(query8,"UPDATE end set entryTime = entryTime, fin = 3 where idx_rn = %d",run);
403         if (mysql_real_query (db, query8, strlen(query8)))
404         {
405             printf("updatefailed query: %s",query8);
406             return(0);
407         }else{
408             return(1);
409         }
410         break;
```

```

411 case 4:
412     sprintf(query9,"UPDATE end set entryTime = entryTime, fin = 0 where idx_rn = %d",run);
413     if (mysql_real_query (db, query9, strlen(query9)))
414     {
415         printf("update failed query: %s",query9);
416         return(0);
417     }else{
418         return(1);
419     }
420     break;
421 default:
422     printf("BAD STUFF");
423     return(0);
424 }
425
426 }
```

4.3.2 Variable Documentation

4.3.2.1 char **buff[255]**

Definition at line 16 of file insTrgCnt.h.

4.3.2.2 int **count = 0**

Definition at line 15 of file insTrgCnt.h.

4.3.2.3 int **fileSequence**

Definition at line 13 of file insTrgCnt.h.

4.3.2.4 int **fileStream**

Definition at line 13 of file insTrgCnt.h.

4.3.2.5 int **idx_trigger**

Definition at line 13 of file insTrgCnt.h.

4.3.2.6 char* **name**

Definition at line 14 of file insTrgCnt.h.

4.3.2.7 int **offlineBit**

Definition at line 13 of file insTrgCnt.h.

4.3.2.8 MYSQL_RES* **res**

Definition at line 11 of file insTrgCnt.h.

4.3.2.9 MYSQL_RES * res2

Definition at line 11 of file insTrgCnt.h.

4.3.2.10 MYSQL_ROW row

Definition at line 12 of file insTrgCnt.h.

4.3.2.11 int tCount

Definition at line 13 of file insTrgCnt.h.

4.3.2.12 int threshVersion

Definition at line 13 of file insTrgCnt.h.

4.3.2.13 int trgVersion

Definition at line 13 of file insTrgCnt.h.

4.4 N_insTrgCnt.cc File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include <time.h>
#include <iostream>
#include "insTrgCnt.h"
```

Functions

- int **main** (int argc, char *argv[])
- int **tCounter** (int run, MYSQL *db_conditions, MYSQL *db_runlog, MYSQL *db_trigger, int two)
- int **rundone** (int run, MYSQL *db_conditions, MYSQL *db_runlog)
- int **getX** (MYSQL *db, char *query)
- int **updateFin** (MYSQL *db, int run, int val)
- int **checkTags** (MYSQL *db, int run)
- void **cleanUp** (MYSQL *db_conditions, MYSQL *db_runlog, MYSQL *db_trigger, int value, int run)

4.4.1 Function Documentation

4.4.1.1 int checkTags (MYSQL * db, int run)

Definition at line 428 of file N_insTrgCnt.cc.

```
429 {
430     int event = 0;
431     int runTag=0;
432     char query5[255];
433     char query6[255];
434     // MYSQL_RES *res;
435     //MYSQL_ROW row;
436
437     sprintf (query5,"select run from daqEventTag where run = %d",run);
438     sprintf (query6,"select run from daqRunTag where run = %d",run);
439
440     if (mysql_real_query (db, query5, strlen(query5)))
441     {
442         printf("check 1 query failed %s\n",query5);
443         event=0;
444     }else{
445         res = mysql_store_result(db);
446         row = mysql_fetch_row(res);
447         if(row)
448             event=1;
449     }
450 }
451 mysql_free_result(res);
452
453
454 if (mysql_real_query (db, query6, strlen(query6)))
455 {
```

```

456     int erno = mysql_errno(db);
457     const char* err = mysql_error(db);
458     printf("check 2 query failed %s \t%d\t%s\n",query6,erno,err);
459     runTag=0;
460 }else{
461
462
463     res = mysql_store_result(db);
464     row = mysql_fetch_row(res);
465     if(row)
466         runTag=1;
467     }
468
469     mysql_free_result(res);
470
471 //  printf("runtag = %d and event =%d, run = %d \n",runTag,event,run);
472 if ((runTag && event))
473     return 1;
474 else{ return 0;}
475 }
```

4.4.1.2 void cleanUp (MYSQL * db_conditions, MYSQL * db_runlog, MYSQL * db_trigger, int value, int run)

Definition at line 477 of file N_insTrgCnt.cc.

```

478 {
479
480     switch (value)
481     {
482         case 1:
483             printf("%d: \t All runs are finished \n", time(NULL));
484             break;
485         case 2:
486             printf("%d: \t end.fin updated to 3, run %d will be rechecked\n", time(NULL),run);
487             break;
488         case 3:
489             printf("%d: \t end.fin updated to 1, with triggerCount for run %d\n",time(NULL),run);
490             break;
491         case 4:
492             printf("%d: \t end.fin updated to 2 for run %d\n",time(NULL),run);
493             break;
494         case 5:
495             printf("%d: \t end.fin update to 1,with NO triggerCount update for run %d\n",time(NULL),run);
496             break;
497         case 6:
498             printf("%d: \t General Query Error ^^^^^^message^^^^^ %d\n",time(NULL),run);
499             break;
500         case 7:
501             printf("%d: \t end.fin RESET to 0 %d\n",time(NULL),run);
502             break;
503         case 8:
504             printf("%d: \t end.fin updated to 2, No recorded RUNS, No trigger Updates for run %d\n",time(NULL));
505             break;
506         default:
507             printf("BAD STUFF");
508     }
509
510     mysql_close(db_conditions);
511     mysql_close(db_runlog);
512     mysql_close(db_trigger);
513
514     exit(1);
515 }
```

```
516 }
```

4.4.1.3 int getX (MYSQL *db, char *query)

Definition at line 349 of file N_insTrgCnt.cc.

```
350 {
351     // MYSQL_RES *res;
352     //MYSQL_ROW row;
353     int x;
354     if(mysql_real_query(db, query, strlen(query))) {
355         printf("Error executing query: %s\n",query);
356         return(0);
357     }
358     res = mysql_store_result(db);
359     if(res == NULL) {
360         printf("Error storing (%s) for query %s\n",mysql_error(db),query);
361         return(0);
362     }
363     row = mysql_fetch_row(res);
364     if(row){
365         x= atoi(row[0]);
366     }else{ return 0; }
367     mysql_free_result(res);
368
369     return x;
370 }
```

4.4.1.4 int main (int argc, char *argv[])

Definition at line 38 of file N_insTrgCnt.cc.

```
39 {
40
41 //    if(argc != 2) {
42 //        printf("%s <run>\n",argv[0]);
43 //        return(0);
44 //    }
45
46 //    int run = atoi(argv[1]);
47 //    if(run == 0) {
48 //        printf("Invalid run %s\n",argv[1]);
49 //        return(0);
50 //    }
51
52 static MYSQL db_conditions;
53 static MYSQL db_runlog;
54 static MYSQL db_trigger;
55
56
57 char query[255];
58 char query2[255];
59 char query3[255];
60 char requery[255];
61
62 char buff2[255];
63 char new_trig[35];
64 char queryTime[255];
65
66 int eVal, boo;
67
```

```

68     unsigned int field_count;
69
70     if(!mysql_real_connect(&db_conditions,
71                           "onlsun1",
72                           "jm1",
73                           NULL,
74                           "Conditions_rts",
75                           3501,
76                           NULL,
77                           0)) {
78         printf("Can't connect to Conditions_rts\n");
79         return(0);
80     }
81
82     if(!mysql_real_connect(&db_runlog,
83                           "onlsun1",
84                           "jm1",
85                           NULL,
86                           "RunLog_daq",
87                           3503,
88                           NULL,
89                           0)) {
90         printf("Can't connect to RunLog_daq\n");
91         mysql_close(&db_conditions);
92         return(0);
93     }
94
95     if(!mysql_real_connect(&db_trigger,
96                           "db01",
97                           "deph",
98                           NULL,
99                           "trigger",
100                          3316,
101                          NULL,
102                          0)) {
103         printf("Can't Connect to trigger\n");
104         mysql_close(&db_conditions);
105         mysql_close(&db_runlog);
106         return(0);
107     }
108 }
109
110 sprintf(query,"SELECT idx_rn from end where fin = 0 limit 1");
111 int runChecked = getX(&db_conditions, query);
112
113 if(!runChecked)
114 {
115     sprintf(requery,"SELECT idx_rn from end where fin = 3 limit 1");
116     int runReChecked = getX(&db_conditions, requery);
117     if(!runReChecked)
118     {
119         cleanUp(&db_conditions,&db_runlog,&db_trigger,1,runChecked);
120     }else{
121         int updated2 = updateFin(&db_conditions, runReChecked,4);
122         cleanUp(&db_conditions,&db_runlog,&db_trigger,7,runReChecked);
123     }
124 }else{
125
126     int insRun = r undone(runChecked, &db_conditions, &db_runlog);
127     if(!insRun)
128     {
129         sprintf(queryTime,"SELECT idxoptime+28800 from end where idx_rn =%d",runChecked);
130         if (mysql_real_query (&db_conditions, queryTime, strlen(queryTime)))
131         {
132             printf("THIS query failed: %s\n%s\n",query,mysql_error(&db_conditions));
133             cleanUp(&db_conditions,&db_runlog,&db_trigger,6,runChecked);
134         }

```

```

135         res=mysql_store_result(&db_conditions);
136         if (res)
137         {
138             row= mysql_fetch_row (res);
139             if(time(NULL)>atoi(row[0]))
140             {
141                 int t = updateFin(&db_conditions, runChecked, 2);
142                 if(t)
143                 {
144                     boo = tCounter(runChecked, &db_conditions, &db_runlog, &db_trigger,t);
145                     cleanUp(&db_conditions,&db_runlog,&db_trigger,4, runChecked);
146                 }
147             }else{
148                 int holdIt = updateFin(&db_conditions, runChecked, 3);
149                 if(holdIt)
150                 {
151                     cleanUp(&db_conditions,&db_runlog,&db_trigger,2, runChecked);
152                 }
153             }
154         }
155     }
156 }else{
157     if(checkTags(&db_runlog, runChecked))
158     {
159         eVal = tCounter(runChecked, &db_conditions,&db_runlog,&db_trigger,0);
160         cleanUp(&db_conditions,&db_runlog,&db_trigger,eVal,runChecked);
161     }else{
162         int skipIt = updateFin(&db_conditions, runChecked, 2);
163         if(skipIt)
164         {
165             cleanUp(&db_conditions,&db_runlog, &db_trigger,8,runChecked);
166         }
167     }
168 }
169 }
170 }
```

4.4.1.5 int rundone (int *run*, MYSQL * *db_conditions*, MYSQL * *db_runlog*)

Definition at line 267 of file N_insTrgCnt.cc.

```

268 {
269
270 static char *dbname = "Conditions_rts";
271 static char *cquery = "select offlineBit, name from triggers where idx_rn=";
272
273     int dbdone=0;
274
275     int nevts;
276     int nevts_et;
277     int end_testcount;
278     int nevbs;
279     int dest;
280     int ntags;
281
282     char query[255];
283
284 // check config to get list of evbs
285 // check end_test run record
286 // get count of events from evbs
287 // get count of events from eventTag
288 sprintf(query, "select count(*) from end where idx_rn=%d",run);
289 end_testcount = getX(db_conditions, query);
290 }
```

```

291     if(end_testcount == 0) {
292         printf("No end_test run record\n");
293         dbdone = 0;
294         goto done;
295     }
296
297     sprintf(query, "select count(*) from nodes where task=20 and idx_rn=%d",run);
298     nevbs = getX(db_conditions, query);
299
300     if(nevbs == 0) {
301         printf("No evbs in run %d.  Dbs as done as they will get.\n",run);
302         dbdone = 1;
303         goto done;
304     }
305
306     // Get the data destination
307
308     sprintf(query, "select destination from run where idx_rn=%d",run);
309     dest = getX(db_conditions, query);
310
311     if((dest != 2) &&
312        (dest != 4) &&
313        (dest != 3)) {
314
315         printf("No db records will be written for this run, run %d (data dest=%d)\n",run,dest);
316         dbdone = 1;
317         goto done;
318     }
319
320     sprintf(query, "select count(*) from daqRunTag where whichtag=1 and run=%d",run);
321     ntags = getX(db_runlog, query);
322
323     if(ntags != nevbs) {
324         printf("The number of run tags for run %d does not match the number of evbs %d vs %d\n", run, ntags,
325             dbdone = 0;
326             goto done;
327     }
328
329     sprintf(query, "select sum(totalNumberOfEvents) from daqRunTag where run=%d\n",run);
330     nevts = getX(db_runlog, query);
331
332     sprintf(query, "select count(*) from daqEventTag where run=%d\n",run);
333     nevts_et = getX(db_runlog, query);
334
335     if(nevts != nevts_et) {
336         printf("Run tag and event tag disagree on number of events in run %d: %d vs %d\n",
337             run, nevts, nevts_et);
338         dbdone = 0;
339         goto done;
340     }
341
342     dbdone = 1;
343 done:
344
345     printf(dbdone ? "Run DBs are done\n" : "Run DBs are not done\n");
346     return(dbdone);
347 }
```

4.4.1.6 int tCounter (int *run*, MYSQL **db_conditions*, MYSQL **db_runlog*, MYSQL **db_trigger*, int *two*)

Definition at line 172 of file N_insTrgCnt.cc.

```
173 {
```

```

174     int val;
175     int updated;
176     int updated2;
177     char query[255];
178     char query2[255];
179     char query3[255];
180
181
182     sprintf(query,"SELECT name, trgVersion, threshVersion, offlineBit, idx_trigger from triggers where
183             if (mysql_real_query (db_conditions, query, strlen(query)))
184             {
185                 printf("THIS query failed: %s\n%s\n",query,mysql_error(db_conditions));
186                 return(6);
187             }else{
188                 res= mysql_store_result(db_conditions);
189                 if (res)
190                 {
191                     while (( row = mysql_fetch_row (res)) !=NULL)
192                     {
193                         for (int i = 0; i< mysql_num_fields(res); i++)
194                         {
195                             name = row[0];
196                             trgVersion = atoi(row[1]);
197                             threshVersion = atoi(row[2]);
198                             offlineBit = atoi(row[3]);
199                             idx_trigger = atoi(row[4]);
200                         }
201                     sprintf(query2,"SELECT count(*), fileStream, fileSequence from daqEventTag where
202                         (void) mysql_real_escape_string (db_runlog,buf,query2,strlen(query2));
203
204                         if (mysql_real_query (db_runlog, buf, strlen(buf)))
205                         {
206                             printf("query2 failed: %s\n(%s)\n",query2,mysql_error(db_runlog));
207                             return(6);
208                         }
209                         res2=mysql_store_result(db_runlog);
210                         if (res2)
211                         {
212                             while ((row = mysql_fetch_row (res2)) !=NULL)
213                             {
214                                 for (int j = 0; j < mysql_num_fields (res2);j++)
215                                 {
216                                     tCount = atoi(row[0]);
217                                     fileStream = atoi(row[1]);
218                                     fileSequence = atoi(row[2]);
219                                 }
220                             sprintf(query3,"INSERT into triggerCount set name = '%s', trgVersion =
221                             if (mysql_real_query (db_trigger, query3, strlen(query3)))
222                             {
223                                 printf("insert failed query = %s\n%s\n",query3,mysql_error(db_trig
224                                 return(6);
225
226                                 }else{
227                                     count++;
228                                 }
229                             }
230                         }else{
231                             printf("INNER query returned nothing\n");
232                             return(6);
233                         }
234                     }
235
236
237             }else{
238                 if(!two){
239                     updated = updateFin(db_conditions, run,1);
240                 }else{

```

```

241             updated = updateFin(db_conditions, run, 2);
242         }
243         if(updated){
244             return(5);
245         }else{
246             printf("\nupdate failed for run %d\n", run);
247             return(6);
248         }
249     }
250     if(!two){
251         updated2 = updateFin(db_conditions, run,1);
252     }else{
253         updated2 = updateFin(db_conditions, run,2);
254     }
255     if(updated2)
256     {
257         printf("triggerCount updated with %d inserts \n",count);
258         return(3);
259     }else{
260         printf("update failed for run %d\n%s\n", run,mysql_error(db_conditions));
261         return(6);
262     }
263 }
264 }
```

4.4.1.7 int updateFin (MYSQL * db, int run, int val)

Definition at line 372 of file N_insTrgCnt.cc.

```

373 {
374     char query4[255];
375     char query7[255];
376     char query8[255];
377     char query9[255];
378
379     switch (val)
380     {
381     case 1:
382         sprintf(query4,"UPDATE end set entryTime = entryTime, fin = 1 where idx_rn = %d",run);
383         if (mysql_real_query (db, query4, strlen(query4)))
384         {
385             printf("updatefailed query: %s",query4);
386             return(0);
387         }else{
388             return(1);
389         }
390         break;
391     case 2:
392         sprintf(query7,"UPDATE end set entryTime = entryTime, fin = 2 where idx_rn = %d",run);
393         if (mysql_real_query (db, query7, strlen(query7)))
394         {
395             printf("updatefailed query: %s",query7);
396             return(0);
397         }else{
398             return(1);
399         }
400         break;
401     case 3:
402         sprintf(query8,"UPDATE end set entryTime = entryTime, fin = 3 where idx_rn = %d",run);
403         if (mysql_real_query (db, query8, strlen(query8)))
404         {
405             printf("updatefailed query: %s",query8);
406             return(0);
407         }else{
```

```
408         return(1);
409     }
410     break;
411 case 4:
412     sprintf(query9,"UPDATE end set entryTime = entryTime, fin = 0 where idx_rn = %d",run);
413     if (mysql_real_query (db, query9, strlen(query9)))
414     {
415         printf("updatefailed query: %s",query9);
416         return(0);
417     }else{
418         return(1);
419     }
420     break;
421 default:
422     printf("BAD STUFF");
423     return(0);
424 }
425
426 }
```

4.5 resync.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include <iostream>
#include "resync.h"
#include "time_keeper.h"
```

Functions

- int **main** (int argc, char *argv[])
- int **rowCmp** (MYSQL_ROW str1, MYSQL_ROW str2, MYSQL_RES *res)
- void **printRow** (MYSQL_ROW row, MYSQL_RES *res, int flag)
- void **printPrimaryKeys** (MYSQL_RES *res)
- void **help** ()
- void **cleanUp** (MYSQL *dbH, MYSQL *dbS, int value)

4.5.1 Function Documentation

4.5.1.1 void **cleanUp** (MYSQL * *dbH*, MYSQL * *dbS*, int *value*)

Definition at line 259 of file resync.c.

```
260 {
261     switch (value)
262     {
263         case 1:
264             printf ("\n\ttables are synchronized \n\n");
265             prn_time();
266             break;
267         case 2:
268             printf ("\n\ttables are NOT synchronized \n\n");
269             prn_time();
270             break;
271         default:
272             printf ("SOMETHING IS VERY WRONG! (cleanUp() switch default) \n");
273     }
274
275     mysql_close(dbH);
276     mysql_close(dbS);
277
278     exit(1);
279 }
```

4.5.1.2 void **help** ()

Definition at line 252 of file resync.c.

```

253 {
254   printf("HELP\n\tresync checks two tables from two different servers (a master and a slave) to \n\t");
255   printf("\nUSAGE:\n\t>resync databasename host1(master) port user table host2(slave) port\nFor example:");
256
257 }

```

4.5.1.3 int main (int argc, char * argv[])

Definition at line 16 of file resync.c.

```

17 {
18   if ((argc < 2 || strcmp(argv[1],"--help")==0 ))
19   {
20     help();
21     return(1);
22   }
23   if (argc < 8)
24   {
25     printf("usage is incorrect - use --help for correct usage\n");
26     return(1);
27   }
28
29   char * pass = NULL;
30   char *db = argv[1];
31   char *mhost = argv[2];
32   int mport = atoi(argv[3]);
33   char *user = argv[4];
34   char *table = argv[5];
35   char *shost = argv[6];
36   int sport = atoi(argv[7]);
37
38   if ( !strcmp(argv[7],"3336" ))
39   {
40     pass = "mirr0r";
41   }
42 }
43
44   char queryOne[40];
45   char queryTwo[255];
46   char queryAdd[155];
47   char orderBy[155];
48   char orderAdd[155];
49   int val = 1;
50
51   printf("\nFOR TABLE %s\n",table);
52   if (!mysql_real_connect(&dBH,mhost,user,pass,db,mport,NULL,0))
53   {
54     printf("can't connect to Master %s %s %s\n",db,mhost,mport);
55   }
56
57   if (!mysql_real_connect(&dBs,shost,user,pass,db,sport,NULL,0))
58   {
59     printf("can't connect to Slave %s %s %s\n",db,shost,sport);
60   }
61
62   sprintf(queryOne,"SELECT * from %s limit 1",table);
63   if (mysql_real_query (&dBH, queryOne, strlen(queryOne)))
64   {
65     printf("THIS query failed: %s\n%s\n",queryOne,mysql_error(&dBH));
66   }
67
68   res=mysql_store_result(&dBH);
69   int max = mysql_num_fields(res);
70

```

```

71             char* fieldList[max];
72             char* fieldNum[max];
73
74     // initialize to all 0s
75     for( int k =0 ; k<max; k++)
76     {
77         fieldList[k] = 0;
78         fieldNum[k] = 0;
79     }
80 //GET the primary Keys separate them based on type
81     int j =0;
82         field = mysql_fetch_fields (res);
83     for (int i = 0; i<mysql_num_fields(res) ; i++)
84     {
85         field = mysql_fetch_field (res);
86
87         if(IS_PRI_KEY(field->flags)){
88             if(IS_NUM(field->type)){
89                 fieldNum[j] = field->name;
90                 // printf("FIELD NUM NAME IS %s \n",field->name);
91             }else{
92                 // printf("FIELD NAME IS %s \n",field->name);
93                 fieldList[j]= field->name;
94             }
95             j++;
96         }
97     }
98
99 //GET THE NUMBER OF PRIMARY KEYS
100    int count = 0, numbers =0, characters = 0;
101    for (int m = 0; m < max; m++)
102    {
103        if (fieldList[m]) {count++;}
104        if (fieldNum[m]) {count++;}
105    }
106 //strcat queryTwo here
107     sprintf(queryTwo,"SELECT ");
108     sprintf(orderBy, " ORDER BY ");
109
110    for (int n = 0; n < count; n++)
111    {
112        if (n < count-1)
113        {
114            if (fieldList[n]){
115                sprintf(queryAdd,"%s, ",fieldList[n]);
116                strcat(queryTwo,queryAdd);
117            }else{
118                sprintf(queryAdd,"%s, ",fieldNum[n]);
119                sprintf(orderAdd,"%s, ",fieldNum[n]);
120                strcat(orderBy,orderAdd);
121                strcat(queryTwo,queryAdd);
122            }
123        }else{
124            if (fieldList[n]){
125                sprintf(queryAdd,"%s FROM %s", fieldList[n],table);
126                strcat(queryTwo,queryAdd);
127            }else{
128                sprintf(queryAdd,"%s FROM %s", fieldNum[n],table);
129                sprintf(orderAdd,"%s",fieldNum[n]);
130                strcat(orderBy,orderAdd);
131                strcat(queryTwo,queryAdd);
132            }
133        }
134    }
135    strcat(queryTwo,orderBy);
136    // printf("QUERY IS %s\n",queryTwo);
137

```

```

138         //GET ALL KEYS FROM MASTER TABLE
139
140         start_time();
141         if (mysql_real_query (&dbH, queryTwo, strlen(queryTwo)))
142         {
143             printf("THIS query failed: %s\n%s\n",queryTwo,mysql_error(&dbH));
144
145         }else{
146             resM=mysql_store_result(&dbH);
147         }
148
149         if(resM)
150         {
151             printf("\nNUM OF ROWS FROM %s IS = %d\n\n",mhost,mysql_num_rows(resM));
152             //GET ALL KEYS FROM SLAVE TABLE
153             prn_time();
154             if (mysql_real_query (&dbS, queryTwo, strlen(queryTwo)))
155             {
156                 printf("THIS query failed: %s\n%s\n",queryTwo,mysql_error(&dbS));
157             }else{
158
159
160                 resS=mysql_store_result(&dbS);
161
162                 if(resS)
163                 {
164                     printf("\nNUM OF ROWS FROM %s IS = %d\n\n",shost,mysql_num_rows(resS));
165                     prn_time();
166                     printPrimaryKeys(resS);
167
168                     int cmp=0;
169
170                     while(1){
171                         if (cmp<=0)
172                         {
173                             rowM = mysql_fetch_row(resM);
174                         }
175                         if (cmp >=0)
176                         {
177                             rowS = mysql_fetch_row(resS);
178                         }
179                         if (rowM == 0 && rowS == 0) break;
180                         cmp = rowCmp(rowM,rowS,resS);
181                         if (cmp>0)
182                         {
183                             printRow(rowS,resS,1);
184                             printf("is missing from %s\n",mhost);
185                             val=2;
186                         }
187                         if (cmp<0)
188                         {
189                             printRow(rowM, resM,1);
190                             printf("is missing from %s\n",shost);
191                             val=2;
192                         }
193                     }
194                 }else{
195                     printf("NO DATA RETURNED FOR %s\n",table);
196                 }
197             }
198         }
199         cleanUp(&dbH,&dbS,val);
200
201     }
202 }
```

4.5.1.4 void printPrimaryKeys (MYSQL_RES *res)

Definition at line 239 of file resync.c.

```

240 {
241     printf("primary keys are:\n\n");
242     for(int x = 0; x<mysql_num_fields(res); x++)
243     {
244         field2 = mysql_fetch_field(res);
245         if(IS_PRI_KEY(field2->flags))
246         {
247             printf("****%s***\t\t",field2->name);
248         }
249     }
250     printf("\n");
251 }
```

4.5.1.5 void printRow (MYSQL_ROW row, MYSQL_RES *res, int flag)

Definition at line 228 of file resync.c.

```

229 {
230     //PRINT OUT A ROW FROM a RESULT SET
231     int num = mysql_num_fields(res);
232
233 for (int i=0 ; i<mysql_num_fields(res) ; i++)
234     {
235         // printf ("\t%s",field->name );
236         printf ("\t***%s***\n",row[i] );
237     }
238 }
```

4.5.1.6 int rowCmp (MYSQL_ROW str1, MYSQL_ROW str2, MYSQL_RES *res)

Definition at line 205 of file resync.c.

```

206 {
207     if (str1 == 0) return 1;
208     if (str2 == 0) return -1;
209
210     for (int i=0; i<mysql_num_fields(resM);i++)
211     {
212
213         if(IS_NUM(field->type)){
214             float a = atof(str1[i]);
215             float b = atof(str2[i]);
216             float ans = a - b;
217             if (ans) return ans;
218         }else{
219
220             int ans = strcmp(str1[i],str2[i]);
221             if (ans) return ans;
222         }
223     }
224     return 0;
225 }
226 }
```

4.6 resync.h File Reference

Functions

- int [rowCmp](#) (MYSQL_ROW, MYSQL_ROW, MYSQL_RES *)
- void [printRow](#) (MYSQL_ROW, MYSQL_RES *, int)
- void [printPrimaryKeys](#) (MYSQL_RES *)
- void [cleanUp](#) (MYSQL *, MYSQL *, int)
- void [help](#) ()

Variables

- MYSQL [dbH](#)
- MYSQL [dbS](#)
- MYSQL_FIELD * [field](#)
- MYSQL_FIELD * [field2](#)
- MYSQL_RES * [res](#)
- MYSQL_RES * [resM](#)
- MYSQL_RES * [resS](#)
- MYSQL_ROW [rowM](#) = 0 [rowS](#) = 0

4.6.1 Function Documentation

4.6.1.1 void [cleanUp](#) (MYSQL *, MYSQL *, int)

Definition at line 268 of file eventCnts.c.

```

269 {
270     switch (value)
271     {
272         case 1:
273             printf( "\n\ttables are synchronized \n\n");
274             prn_time();
275             break;
276         case 2:
277             printf (" \n\ttables are NOT synchronized \n\n");
278             prn_time();
279             break;
280         default:
281             printf ("SOMETHING IS VERY WRONG! (cleanUp() switch default) \n");
282     }
283     mysql_close(dbH);
284     mysql_close(dbS);
285
286     exit(1);
287 }
```

4.6.1.2 void [help](#) ()

Definition at line 261 of file eventCnts.c.

```

262 {
263   printf("HELP\n\tresync checks two tables from two different servers (a master and a slave) to \n\t1")
264   printf("\nUSAGE:\n\t>resync databaseName host1(master) port user table host2(slave) port\nFor example"
265
266 }

```

4.6.1.3 void printPrimaryKeys (MYSQL_RES *)

Definition at line 248 of file eventCnts.c.

```

249 {
250   printf("primary keys are:\n\n");
251   for(int x = 0; x<mysql_num_fields(res); x++)
252   {
253     field2 = mysql_fetch_field(res);
254     if(IS_PRI_KEY(field2->flags))
255     {
256       printf("****%s***\t\t",field2->name);
257     }
258   }
259   printf("\n");
260 }

```

4.6.1.4 void printRow (MYSQL_ROW, MYSQL_RES *, int)

Definition at line 237 of file eventCnts.c.

```

238 {
239   //PRINT OUT A ROW FROM a RESULT SET
240   int num = mysql_num_fields(res);
241
242   for (int i=0 ; i<mysql_num_fields(res) ; i++)
243   {
244     // printf("\t%s",field->name );
245     printf("\t***%s***\n",row[i] );
246   }
247 }

```

4.6.1.5 int rowCmp (MYSQL_ROW, MYSQL_ROW, MYSQL_RES *)

Definition at line 214 of file eventCnts.c.

```

215 {
216   if (str1 == 0) return 1;
217   if (str2 == 0) return -1;
218
219   for (int i=0; i<mysql_num_fields(resM);i++)
220   {
221     if(IS_NUM(field->type)){
222       float a = atof(str1[i]);
223       float b = atof(str2[i]);
224       float ans = a - b;
225       if (ans) return ans;
226     }else{
227       int ans = strcmp(str1[i],str2[i]);

```

```
230         if (ans) return ans;
231     }
232   }
233   return 0;
234
235 }
```

4.6.2 Variable Documentation

4.6.2.1 MYSQL dbH [static]

Definition at line 3 of file resync.h.

4.6.2.2 MYSQL dbS [static]

Definition at line 3 of file resync.h.

4.6.2.3 MYSQL_FIELD* field

Definition at line 5 of file resync.h.

4.6.2.4 MYSQL_FIELD * field2

Definition at line 5 of file resync.h.

4.6.2.5 MYSQL_RES* res

Definition at line 6 of file resync.h.

4.6.2.6 MYSQL_RES * resM

Definition at line 6 of file resync.h.

4.6.2.7 MYSQL_RES * resS

Definition at line 6 of file resync.h.

4.6.2.8 MYSQL_ROW rowM = 0 rowS = 0

Definition at line 7 of file resync.h.

4.7 syncCol.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include <iostream>
#include "syncCol.h"
#include "time_keeper.h"
```

Defines

- #define FILENAME "/star/u/deph/src/out.put"

Functions

- int main (int argc, char *argv[])
- int rowCmp (MYSQL_ROW str1, MYSQL_ROW str2, MYSQL_RES *res)
- void printRow (MYSQL_ROW row, MYSQL_RES *res, int flag, FILE *fptr)
- void printPrimaryKeys (MYSQL_RES *res)
- void help ()
- void cleanUp (MYSQL *dbH, MYSQL *dbS, int value)

4.7.1 Define Documentation

4.7.1.1 #define FILENAME "/star/u/deph/src/out.put"

Definition at line 18 of file syncCol.c.

4.7.2 Function Documentation

4.7.2.1 void cleanUp (MYSQL * dbH, MYSQL * dbS, int value)

Definition at line 319 of file syncCol.c.

```
320 {
321     switch (value)
322     {
323         case 1:
324             printf( "\n\ttables are synchronized \n\n");
325             prn_time();
326             break;
327         case 2:
328             printf (" \n\ttables are NOT synchonized \n\n");
329             prn_time();
330             break;
331         default:
332             printf ("SOMETHING IS VERY WRONG! (cleanUp() switch default) \n");
333     }
334 }
```

```

335     mysql_close(dbH);
336     mysql_close(dbS);
337
338
339     exit(1);
340 }
```

4.7.2.2 void help()

Definition at line 312 of file syncCol.c.

```

313 {
314     printf("HELP\n\tcolsyn checks two columns to \n\t1) see if they are synchronized \n\t2) identify the
315     printf("\nUSAGE:\n\t>>resync databasename host1(master) port user table host2(slave) port\nFor example
316
317 }
```

4.7.2.3 int main (int argc, char * argv[])

Definition at line 20 of file syncCol.c.

```

21 {
22     if ((argc < 2 || strcmp(argv[1],"--help")==0 ))
23     {
24         help();
25         return(1);
26     }
27     if (argc < 8)
28     {
29         printf("usage is incorrect - use --help for correct usage\n");
30         return(1);
31     }
32
33     char * pass = NULL;
34     char *db = argv[1];
35     char *table1 = argv[2];
36     char *mhost = argv[3];
37     int mport = atoi(argv[4]);
38     char *user = argv[5];
39     char *table2 = argv[6];
40     char *shost = argv[7];
41     int sport = atoi(argv[8]);
42
43     char * table;
44
45     if ( !strcmp(argv[8],"3336" ))
46     {
47         pass = "mirr0r";
48     }
49
50
51     char queryOne[40];
52     char queryTable1[40];
53     char queryTable2[40];
54     char queryTwo[255];
55     char queryThree[255];
56     char queryFour[255];
57     char queryAdd[155];
58     char orderBy[155];
59     char orderAdd[155];
60     int val = 1;
```

```

61
62     FILE *file_ptr;
63     file_ptr = fopen(FILENAME, "w");
64
65
66     printf("\nFOR TABLES %s and %s \n",table1,table2);
67     printf("\n\tWhich columns do you want to compare\n\n");
68
69
70     if (!mysql_real_connect(&dbH,mhost,user,pass,db,mport,NULL,0))
71     {
72         printf("can't connect to Master %s %s %s\n",db,mhost,mport);
73     }
74
75
76     if (!mysql_real_connect(&dbS,shost,user,pass,db,sport,NULL,0))
77     {
78         printf("can't connect to Slave %s %s %s\n",db,shost,sport);
79     }
80
81
82     sprintf(queryOne,"SELECT * from %s limit 1",table1);
83     sprintf(queryTwo,"SELECT * from %s limit 1",table2);
84     if (mysql_real_query (&dbH, queryOne, strlen(queryOne)))
85     {
86         printf("THIS query failed: %s\n%s\n",queryOne,mysql_error(&dbH));
87     }
88
89     if (mysql_real_query (&dbS, queryTwo, strlen(queryTwo)))
90     {
91         printf("THIS query failed: %s\n%s\n",queryOne,mysql_error(&dbH));
92     }
93
94     res=mysql_store_result(&dbH);
95     res2=mysql_store_result(&dbS);
96
97     printf("%s columns are\n",table1);
98
99     field1 = mysql_fetch_fields (res);
100    for (int i = 0; i<mysql_num_fields(res) ; i++)
101    {
102        field1 = mysql_fetch_field (res);
103        printf("FIELD NAME IS %s \n",field1->name);
104    }
105
106    printf("\n%s columns are\n",table2);
107
108    field2 = mysql_fetch_fields (res2);
109    for (int i = 0; i<mysql_num_fields(res2) ; i++)
110    {
111        field2 = mysql_fetch_field (res2);
112        printf("FIELD NAME IS %s \n",field2->name);
113    }
114
115    int max = mysql_num_fields(res);
116    char* fieldList[max];
117    char * fieldNum[max];
118
119    // initialize to all 0s
120    for( int k =0 ; k<max; k++)
121    {
122        fieldList[k] = 0;fieldNum[k] = 0;
123    }
124
125    printf("MAX IS %d\n",max);
126    printf("input each column <<return>> (one column per line) then type \"done\" <<return>> \n");
127    char * x;

```

```

128     char s[100];
129
130     int cnt = 0;
131     char *d ="done";
132
133     printf( "+++++\n");
134
135     while(1)
136     {
137
138         fgets(s,100,stdin);
139         s[strlen(s)-1]= '\0';
140         // printf("%s\n",s);
141         // printf("\n%s,%s\n",d,s);
142
143         if(strcmp(d,s))
144         {
145             x = (char*) malloc(strlen(s)+1);
146             strcpy(x,s);
147             fieldList[cnt] = x;
148
149             cnt++;
150         }else{
151             break;
152         }
153     }
154     x=0;
155     free(x);
156     printf("you inputed: ");
157     for (int i = 0; i < cnt ; i++)
158     {
159         printf("%s\n",fieldList[i]);
160     }
161
162 //strcat queryTwo here
163     sprintf(queryThree,"SELECT distinct ");
164     sprintf(queryFour,"SELECT distinct ");
165     sprintf(orderBy, " ORDER BY ");
166
167     for (int n = 0; n < cnt; n++)
168     {
169         if (n < cnt-1)
170         {
171             if (fieldList[n]){
172                 sprintf(queryAdd,"%s, ",fieldList[n]);
173                 strcat(queryThree,queryAdd);
174                 strcat(queryFour,queryAdd);
175                 sprintf(orderAdd,"%s, ",fieldList[n]);
176                 strcat(orderBy,orderAdd);
177             }
178         }else{
179             if (fieldList[n]){
180                 sprintf(queryAdd,"%s FROM ", fieldList[n]);
181                 strcat(queryThree,queryAdd);
182                 strcat(queryFour,queryAdd);
183
184                 sprintf(orderAdd,"%s ",fieldList[n]);
185                 strcat(orderBy,orderAdd);
186             }
187
188             // strcat(queryThree,queryAdd);
189         }
190     }
191     sprintf(queryTable1,"%s where fileStream > 99 ",table1);
192     sprintf(queryTable2,"%s ",table2);
193     strcat(queryThree,queryTable1);
194     strcat(queryFour, queryTable2);

```

```

195     strcat(queryThree,orderBy);
196     strcat(queryFour, orderBy);
197     printf("QUERY ONE IS %s\n",queryThree);
198     printf("QUERY TWO IS %s\n",queryFour);
199
200     //GET ALL KEYS FROM MASTER TABLE
201
202     start_time();
203     if (mysql_real_query (&dbH, queryThree, strlen(queryThree)))
204     {
205         printf ("THIS query failed: %s\n%s\n",queryThree,mysql_error(&dbH));
206     }else{
207         resM=mysql_store_result(&dbH);
208     }
209
210     if(resM)
211     {
212         printf("\nNUM OF ROWS FROM %s IS = %d\n\n",table1,mysql_num_rows(resM));
213         //GET ALL KEYS FROM SLAVE TABLE
214         prn_time();
215         if (mysql_real_query (&dbS, queryFour, strlen(queryFour)))
216         {
217             printf("THIS query failed: %s\n%s\n",queryFour,mysql_error(&dbS));
218         }else{
219
220             resS=mysql_store_result(&dbS);
221
222             if(resS)
223             {
224                 printf("\nNUM OF ROWS FROM %s IS = %d\n\n",table2,mysql_num_rows(resS));
225                 prn_time();
226                 // printPrimaryKeys(resS);
227
228                 int cmp=0;
229
230                 while(1){
231                     if (cmp<=0)
232                     {
233                         rowM = mysql_fetch_row(resM);
234                     }
235                     if (cmp >=0)
236                     {
237                         rowS = mysql_fetch_row(resS);
238                     }
239                     if (rowM == 0 && rowS == 0) break;
240                     cmp = rowCmp(rowM,rowS,resS);
241                     if (cmp>0)
242                     {
243                         printRow(rowS,resS,1,file_ptr);
244                         fprintf(file_ptr,"is missing from %s\n",table1);
245                         val=2;
246                     }
247                     if (cmp<0)
248                     {
249                         printRow(rowM, resM,1,file_ptr);
250                         fprintf(file_ptr,"is missing from %s\n",table2);
251                         val=2;
252                     }
253                 }else{
254                     printf("NO DATA RETURNED FOR %s\n",table);
255                 }
256             }
257         }
258     }
259     fclose(file_ptr);
260     cleanUp(&dbH,&dbS,val);
261 }
```

4.7.2.4 void printPrimaryKeys (MYSQL_RES *res)

Definition at line 299 of file syncCol.c.

```

300 {
301     printf("primary keys are:\n\n");
302     for(int x = 0; x<mysql_num_fields(res); x++)
303     {
304         field2 = mysql_fetch_field(res);
305         if(IS_PRI_KEY(field2->flags))
306         {
307             printf("****%s***\t\t",field2->name);
308         }
309     }
310     printf("\n");
311 }
```

4.7.2.5 void printRow (MYSQL_ROW row, MYSQL_RES *res, int flag, FILE *fptr)

Definition at line 288 of file syncCol.c.

```

289 {
290     //PRINT OUT A ROW FROM a RESULT SET
291     int num = mysql_num_fields(res);
292
293 for (int i=0 ; i<mysql_num_fields(res) ; i++)
294     {
295         // printf("\t%s",field->name );
296         fprintf(fptr,"\t***%s***\n",row[i] );
297     }
298 }
```

4.7.2.6 int rowCmp (MYSQL_ROW str1, MYSQL_ROW str2, MYSQL_RES *res)

Definition at line 264 of file syncCol.c.

```

265 {
266     if (str1 == 0) return 1;
267     if (str2 == 0) return -1;
268
269     for (int i=0; i<mysql_num_fields(res);i++)
270     {
271
272         if(IS_NUM(field1->type)){
273
274             float a = atof(str1[i]);
275             float b = atof(str2[i]);
276             float ans = a - b;
277             if (ans) return ans;
278         }else{
279
280             int ans = strcmp(str1[i],str2[i]);
281             if (ans) return ans;
282         }
283     }
284     return 0;
285 }
286 }
```

4.8 syncCol.h File Reference

Functions

- int [rowCmp](#) (MYSQL_ROW, MYSQL_ROW, MYSQL_RES *)
- void [printRow](#) (MYSQL_ROW, MYSQL_RES *, int, FILE *)
- void [printPrimaryKeys](#) (MYSQL_RES *)
- void [cleanUp](#) (MYSQL *, MYSQL *, int)
- void [help](#) ()

Variables

- MYSQL [dbH](#)
- MYSQL [dbS](#)
- MYSQL_FIELD * [field](#)
- MYSQL_FIELD * [field1](#)
- MYSQL_FIELD * [field2](#)
- MYSQL_RES * [res](#)
- MYSQL_RES * [res2](#)
- MYSQL_RES * [resS](#)
- MYSQL_RES * [resM](#)
- MYSQL_ROW [rowM](#) = 0 [rowS](#) = 0

4.8.1 Function Documentation

4.8.1.1 void [cleanUp](#) (MYSQL *, MYSQL *, int)

Definition at line 268 of file eventCnts.c.

```

269 {
270     switch (value)
271     {
272         case 1:
273             printf ("\n\ttables are synchronized \n\n");
274             prn_time();
275             break;
276         case 2:
277             printf ("\n\ttables are NOT synchronized \n\n");
278             prn_time();
279             break;
280         default:
281             printf ("SOMETHING IS VERY WRONG! (cleanUp() switch default) \n");
282     }
283
284     mysql_close(dbH);
285     mysql_close(dbS);
286
287     exit(1);
288 }
```

4.8.1.2 void help()

Definition at line 261 of file eventCnts.c.

```
262 {
263     printf("HELP\n\tresync checks two tables from two different servers (a master and a slave) to \n\tt1)
264     printf("\nUSAGE:\n\t>resync databasename host1(master) port user table host2(slave) port\nFor example
265
266 }
```

4.8.1.3 void printPrimaryKeys (MYSQL_RES *)

Definition at line 248 of file eventCnts.c.

```
249 {
250     printf("primary keys are:\n\n");
251     for(int x = 0; x<mysql_num_fields(res); x++)
252     {
253         field2 = mysql_fetch_field(res);
254         if(IS_PRI_KEY(field2->flags))
255         {
256             printf("****%s***\t\t",field2->name);
257         }
258     }
259     printf("\n");
260 }
```

4.8.1.4 void printRow (MYSQL_ROW, MYSQL_RES *, int, FILE *)

Definition at line 288 of file syncCol.c.

```
289 {
290     //PRINT OUT A ROW FROM a RESULT SET
291     int num = mysql_num_fields(res);
292
293 for (int i=0 ; i<mysql_num_fields(res) ; i++)
294     {
295         // printf("\t%s",field->name );
296         fprintf(fptr,"\\t***%s***\n",row[i] );
297     }
298 }
```

4.8.1.5 int rowCmp (MYSQL_ROW, MYSQL_ROW, MYSQL_RES *)

Definition at line 214 of file eventCnts.c.

```
215 {
216     if (str1 == 0) return 1;
217     if (str2 == 0) return -1;
218
219     for (int i=0; i<mysql_num_fields(resM);i++)
220     {
221
222         if(IS_NUM(field->type)){
223             float a = atof(str1[i]);
224             float b = atof(str2[i]);
225
226             if(a < b)
227                 return -1;
228             else if(a > b)
229                 return 1;
230             else
231                 continue;
232         }
233
234         if(strcmp(field->name,str1[i])>0)
235             return 1;
236         else if(strcmp(field->name,str1[i])<0)
237             return -1;
238     }
239
240     if(mysql_num_fields(resM)<mysql_num_fields(res))
241         return -1;
242     else if(mysql_num_fields(resM)>mysql_num_fields(res))
243         return 1;
244
245     return 0;
246 }
```

```
225     float ans = a - b;
226     if (ans) return ans;
227 }else{
228
229     int ans = strcmp(str1[i],str2[i]);
230     if (ans) return ans;
231 }
232 }
233 return 0;
234
235 }
```

4.8.2 Variable Documentation

4.8.2.1 MYSQL **dBH** [static]

Definition at line 3 of file syncCol.h.

4.8.2.2 MYSQL **dBs** [static]

Definition at line 3 of file syncCol.h.

4.8.2.3 MYSQL_FIELD* **field**

Definition at line 5 of file syncCol.h.

4.8.2.4 MYSQL_FIELD * **field1**

Definition at line 5 of file syncCol.h.

4.8.2.5 MYSQL_FIELD * **field2**

Definition at line 5 of file syncCol.h.

4.8.2.6 MYSQL_RES* **res**

Definition at line 6 of file syncCol.h.

4.8.2.7 MYSQL_RES * **res2**

Definition at line 6 of file syncCol.h.

4.8.2.8 MYSQL_RES * **resM**

Definition at line 6 of file syncCol.h.

4.8.2.9 MYSQL_RES * **resS**

Definition at line 6 of file syncCol.h.

4.8.2.10 MYSQL_ROW rowM = 0 rowS = 0

Definition at line 7 of file syncCol.h.

4.9 test.c File Reference

```
#include <mysql.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Functions

- int **main** (int argc, char *argv[])

4.9.1 Function Documentation

4.9.1.1 int main (int *argc*, char * *argv*[])

Definition at line 6 of file test.c.

```
7 {
8     static MYSQL db;
9     MYSQL_RES *res;
10    MYSQL_ROW row;
11    MYSQL_FIELD *field;
12    unsigne long col_len;
13
14    if(!mysql_real_connect(&db,
15                          "onlsun1",
16                          "stardb",
17                          NULL,
18                          "RunLog",
19                          3501,
20                          NULL,
21                          0)) {
22        printf("Error: %s\n",mysql_error(&db));
23    }
24
25    char *q = "select a.runNumber, a.name, a.offlineTriggerID as offlineTrgId,format(a.prescale,1)
26 mysql_real_query(&db, q, strlen(q));
27
28    res = mysql_store_result(&db);
29    if(!res) printf("Error? %s (%s)\n", q, mysql_error(&db));
30
31    mysql_field_seek (res, 0);
32
33    for (unsigned int i; i < mysql_num_fields(res); i++)
34    {
35        field = mysql_fetch_field (res)
36        col_len = strlen (field->name);
37        if (col_len < field->max_length)
38            col_len = field->max_length;
39        if (col_len < 4 && !IS_NOT_NULL (field->flags))
40            col_len = 4; /* 4 = length of the word "NULL" */
41        field->max_length = col_len; /* reset column info */
42    }
43
44    row = mysql_fetch_row(res);
45
46
47    if(row)
```

```
49          // printf("last run is %d\n",atoi(row[0]));
50          printf("last run is %d\n",atoi(row[0]));
51
52      else
53          printf("Error? %s\n",mysql_error(&db));
54      mysql_close(&db);
55 }
```

4.10 time_keeper.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

Classes

- struct [time_keeper](#)

Defines

- #define [MAXSTRING](#) 100

Functions

- void [start_time](#) (void)
- double [prn_time](#) (void)

Variables

- [time_keeper](#) tk

4.10.1 Define Documentation

4.10.1.1 #define MAXSTRING 100

Definition at line 10 of file time_keeper.c.

4.10.2 Function Documentation

4.10.2.1 double prn_time (void)

Definition at line 25 of file time_keeper.c.

```
26 {
27     char s1[MAXSTRING], s2[MAXSTRING];
28     int field_width, n1,n2;
29     double clocks_per_second = (double) CLOCKS_PER_SEC, user_time, real_time;
30
31     user_time = (clock() - tk.save_clock) / clocks_per_second;
32     real_time = difftime(time(NULL), tk.save_time);
33
34     tk.save_clock = clock();
35     tk.save_time = time(NULL);
36
37     n1 = sprintf(s1, "%lf", user_time);
38     n2 = sprintf(s2, "%lf", real_time);
39
40     field_width=(n1>n2) ? n1 : n2 ;
```

```
41     printf("%s%*.1f%s\n%s%*.1f%s\n\n",
42             "User time: ", field_width, user_time, " seconds",
43             "Real time: ", field_width, real_time, " seconds");
44     return user_time;
45 }
46 }
```

4.10.2.2 void start_time (void)

Definition at line 19 of file time_keeper.c.

```
20 {
21     tk.begin_clock = tk.save_clock = clock();
22     tk.begin_time = tk.save_time = time(NULL);
23 }
```

4.10.3 Variable Documentation

4.10.3.1 time_keeper tk [static]

Definition at line 17 of file time_keeper.c.

4.11 time_keeper.h File Reference

Functions

- void [start_time](#) (void)
- void [prn_time](#) (void)

4.11.1 Function Documentation

4.11.1.1 void [prn_time](#) (void)

Definition at line 25 of file time_keeper.c.

```
26 {
27     char s1[MAXSTRING], s2[MAXSTRING];
28     int field_width, n1,n2;
29     double clocks_per_second = (double) CLOCKS_PER_SEC, user_time, real_time;
30
31     user_time = (clock() - tk.save_clock) / clocks_per_second;
32     real_time = difftime(time(NULL), tk.save_time);
33
34     tk.save_clock = clock();
35     tk.save_time = time(NULL);
36
37     n1 = sprintf(s1, "%lf", user_time);
38     n2 = sprintf(s2, "%lf", real_time);
39
40     field_width=(n1>n2) ? n1 : n2 ;
41
42     printf("%s%*.1f%s\n%s%*.1f%s\n\n",
43            "User time: ", field_width, user_time, " seconds",
44            "Real time: ", field_width, real_time, " seconds");
45     return user_time;
46 }
```

4.11.1.2 void [start_time](#) (void)

Definition at line 19 of file time_keeper.c.

```
20 {
21     tk.begin_clock = tk.save_clock = clock();
22     tk.begin_time = tk.save_time = time(NULL);
23 }
```

Index

begin_clock
 time_keeper, 5

begin_time
 time_keeper, 5

boo.c, 7

buff
 insTrgCnt.h, 20

checkTags
 insTrgCnt.h, 14
 N_insTrgCnt.cc, 22

cleanUp
 eventCnts.c, 8
 insTrgCnt.h, 15
 N_insTrgCnt.cc, 23
 resync.c, 31
 resync.h, 36
 syncCol.c, 39
 syncCol.h, 45

count
 insTrgCnt.h, 20

dbH
 resync.h, 38
 syncCol.h, 47

dbS
 resync.h, 38
 syncCol.h, 47

eventCnts.c, 8

eventCnts.c
 cleanUp, 8
 help, 8
 main, 9
 printPrimaryKeys, 12
 printRow, 12
 rowCmp, 12

field
 resync.h, 38
 syncCol.h, 47

field1
 syncCol.h, 47

field2
 resync.h, 38
 syncCol.h, 47

FILENAME
 syncCol.c, 39

fileSequence
 insTrgCnt.h, 20

fileStream
 insTrgCnt.h, 20

getX
 insTrgCnt.h, 16
 N_insTrgCnt.cc, 24

help
 eventCnts.c, 8
 resync.c, 31
 resync.h, 36
 syncCol.c, 40
 syncCol.h, 45

idx_trigger
 insTrgCnt.h, 20

insTrgCnt.h, 14

insTrgCnt.h
 buff, 20
 checkTags, 14
 cleanUp, 15
 count, 20
 fileSequence, 20
 fileStream, 20
 getX, 16
 idx_trigger, 20
 name, 20
 offlineBit, 20
 res, 20
 res2, 20
 row, 21
 rundone, 16
 tCount, 21
 tCounter, 17
 threshVersion, 21
 trgVersion, 21
 updateFin, 19

main
 eventCnts.c, 9
 N_insTrgCnt.cc, 24
 resync.c, 32

syncCol.c, 40
test.c, 49
MAXSTRING
time_keeper.c, 51

N_insTrgCnt.cc, 22
N_insTrgCnt.cc
checkTags, 22
cleanUp, 23
getX, 24
main, 24
rundone, 26
tCounter, 27
updateFin, 29

name
insTrgCnt.h, 20

offlineBit
insTrgCnt.h, 20

printPrimaryKeys
eventCnts.c, 12
resync.c, 34
resync.h, 37
syncCol.c, 43
syncCol.h, 46

printRow
eventCnts.c, 12
resync.c, 35
resync.h, 37
syncCol.c, 44
syncCol.h, 46

prn_time
time_keeper.c, 51
time_keeper.h, 53

res
insTrgCnt.h, 20
resync.h, 38
syncCol.h, 47

res2
insTrgCnt.h, 20
syncCol.h, 47

resM
resync.h, 38
syncCol.h, 47

resS
resync.h, 38
syncCol.h, 47

resync.c, 31
cleanUp, 31
help, 31
main, 32
printPrimaryKeys, 34

printRow, 35
rowCmp, 35

resync.h, 36
cleanUp, 36
dBH, 38
dBs, 38
field, 38
field2, 38
help, 36
printPrimaryKeys, 37
printRow, 37
res, 38
resM, 38
resS, 38
rowCmp, 37
rowM, 38

row
insTrgCnt.h, 21

rowCmp
eventCnts.c, 12
resync.c, 35
resync.h, 37
syncCol.c, 44
syncCol.h, 46

rowM
resync.h, 38
syncCol.h, 47

rundone
insTrgCnt.h, 16
N_insTrgCnt.cc, 26

save_clock
time_keeper, 5

save_time
time_keeper, 5

start_time
time_keeper.c, 52
time_keeper.h, 53

syncCol.c, 39
syncCol.c
cleanUp, 39
FILENAME, 39
help, 40
main, 40
printPrimaryKeys, 43
printRow, 44
rowCmp, 44

syncCol.h, 45
syncCol.h
cleanUp, 45
dBH, 47
dBs, 47
field, 47
field1, 47

field2, 47
help, 45
printPrimaryKeys, 46
printRow, 46
res, 47
res2, 47
resM, 47
resS, 47
rowCmp, 46
rowM, 47

tCount
 insTrgCnt.h, 21

tCounter
 insTrgCnt.h, 17
 N_insTrgCnt.cc, 27

test.c, 49
 main, 49

threshVersion
 insTrgCnt.h, 21

time_keeper, 5
 begin_clock, 5
 begin_time, 5
 save_clock, 5
 save_time, 5

time_keeper.c, 51
 MAXSTRING, 51
 prn_time, 51
 start_time, 52
 tk, 52

time_keeper.h, 53
 prn_time, 53
 start_time, 53

tk
 time_keeper.c, 52

trgVersion
 insTrgCnt.h, 21

updateFin
 insTrgCnt.h, 19
 N_insTrgCnt.cc, 29